Name: Harsh Shivankar
PRN: 21070521028
Department: CSE
Division: A

Symbiosis Institute of Technology
Department of Applied Science

Generative AI
CA-2

Computer Science and Engineering

Batch 2021-2025

Semester - VII

Python Notebook Summaries and Code

Summary of Q1.ipynb

The first notebook simulates 100 savings accounts with random initial balances and random transactions (deposits and withdrawals) over a period of 12 months.

Summary:

- A SavingsAccount class is defined with methods for deposits, withdrawals, and getting balance.
- The simulation generates 100 accounts, each with a unique account ID and an initial random balance.
- For 12 months, each account either receives deposits or makes withdrawals.
- The final balances of all accounts are displayed.

Q1 Code

```
import random
import pandas as pd

# Seed for reproducibility
random.seed(42)

class SavingsAccount:
    def __init__(self, account_id, initial_balance):
        self.account_id = account_id
        self.balance = initial_balance
```

```
        self.transactions = []

    def deposit(self, amount):
        self.balance += amount
        self.transactions.append(('Deposit', amount, self.balance))

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            self.transactions.append(('Withdraw', amount, self.balance))
        else:
            self.transactions.append(('Failed Withdraw', amount, self.balance))

    def get_balance(self):
        return self.balance

    def get_transaction_history(self):
        return self.transactions

# Function to generate random accounts
def generate_accounts(num_accounts, months):
    accounts = []
    for i in range(num_accounts):
        account_id = "ACC{:03d}".format(random.randint(1, 100))
        initial_balance = random.uniform(0, 1000)
        account = SavingsAccount(account_id, initial_balance)
        for _ in range(months):
```

```
            if random.choice([True, False]):
                account.deposit(random.uniform(0, 500))
            else:
                account.withdraw(random.uniform(0, 500))
        accounts.append(account)
    return accounts

# Generate 100 accounts over 12 months
accounts = generate_accounts(100, 12)

# Create a summary of account balances
data = {'Account ID': [acc.account_id for acc in accounts],
        'Final Balance': [acc.get_balance() for acc in accounts]}

df = pd.DataFrame(data)
print(df)
```

Q1 Output

```
   Account ID  Final Balance
0      ACC069     163.567529
1      ACC098     323.428555
...
99     ACC060   12426.587660

[100 rows x 2 columns]
```

Summary of Q5.ipynb

The second notebook generates a dataset of 100 samples, each containing symptoms such as fever, cold, shivering, and weight loss. A function is defined to sort the dataset based on any parameter (e.g., fever).

Summary:

- Random symptom data is generated for 100 samples.
- The symptoms include fever (temperature), cold (yes/no), shivering (yes/no), and weight loss (in kg).
- A function `sort_by_parameter` sorts the data based on any column (e.g., fever).

Q5 Code

```
import pandas as pd
import numpy as np

# Define the number of samples
n_samples = 100

# Create random data for symptoms
np.random.seed(42)  # for reproducibility
data = {
    'fever': np.random.uniform(98, 105, n_samples),  # random temperature values in Fahrenheit
    'cold': np.random.choice([0, 1], n_samples),     # 0 for no cold, 1 for cold
    'shivering': np.random.choice([0, 1], n_samples), # 0 for no shivering, 1 for shivering
```

```
    'weight_loss': np.random.uniform(0, 10, n_samples) # random weight loss in kg
}

# Create a DataFrame
df = pd.DataFrame(data)

# Define a function to sort data based on an input parameter
def sort_by_parameter(df, parameter):
    return df.sort_values(by=parameter)

# Sort by 'fever' as an example
sorted_df = sort_by_parameter(df, 'fever')

# Display the sorted data
print(sorted_df)
```

Q5 Output

```
        fever  cold  shivering  weight_loss
72   98.038655     1          1     8.670723
10   98.144091     0          1     5.487338
...
69  104.908209     0          0     2.935918

[100 rows x 4 columns]
```